# COMP 304 Shellington: Project 1

Due: 11 Nov 2021 midnight

*Didem Unat Fall 2021*

**Notes:** The project can be done **individually** or **teams of 2**. You may discuss the problems with other teams and post questions to the OS discussion forum but the submitted work must be your own work. This assignment is worth 15% of your total grade. Read this document carefully before you start and START EARLY.

**Any material you use from web should be properly cited in your report. Any sort of cheating will be harshly PUNISHED.**

Contact TA: Ismayil Ismayilov (ENG 230)
iismayilov21@ku.edu.tr

## Description

The main part of the project requires development of an interactive Unix-style operating system shell, called **shellington** in C/C++. After executing **shellington**, **shellington** will read both system and user-defined commands from the user. The project has three main parts:

## Part I - Basic Commands

**20 points**

The shell must implement basic commands and support the following:

- Use the skeleton program provided as a starting point for your implementation. The skeleton program reads the next command line, parses and separates it into distinct arguments using blanks as delimiters. You will implement the action that needs to be taken based on the command and its arguments entered to **shellington**. Feel free to modify the command line parser as you wish.
- Commandline inputs should be interpreted as program invocation, which should be done by the shell **fork**ing and **exec**ing the programs as its own child processes.
- The shell must support background execution of programs. An ampersand (&) at the end of the command line indicates that the shell should return the command line prompt immediately after launching that program.
- Use execv() system call (instead of execvp()) to execute UNIX commands (e.g. ls, mkdir, cp, mv, date, gcc) and user programs by the child process.

The descriptions in the book might be useful. You can read Project 1- Unix Shell Part-I in Chapter 3 starting from Page 157 (9th edition).

## Part II - Custom Commands

In this part, you are asked to implement a number of user-defined commands in **shellington**.

### short (10 pts)

The first command is called **short** which takes two arguments, first one is 'set' or 'jump' and second one is the alias (short name) that will be assigned to the current working directory. The command 'short set name' takes the current directory and associates the name with the path of the current directory. When a user enters 'short jump name', the current directory is changed to the directory associated with the name. Overall, this function enables storing directories with short names and jumping to that directory using the short name. One directory can be associated with multiple short names; however, one short name indicates only one directory which can be overwritten later.

### bookmark (15 pts)

The second command is the **bookmark** command. This feature will enable users to bookmark frequently used commands. See the following example to add a command to the bookmarks and execute it.

```
1  myshell> bookmark "cd /home/users/xxx/yyy/zzz"
2  myshell> bookmark "ssh dunat@login.kuacc.ku.edu.tr"
3  myshell> bookmark -l
4      0 "cd /home/users/xxx/yyy/zzz"
5      1 "ssh dunat@lufer.hpc.ku.edu.tr"
6  myshell> bookmark -i 0
7  /home/users/xxx/yyy/zzz>
8  myshell> bookmark -d 0
9  myshell> bookmark -l
10     0 "ssh dunat@lufer.hpc.ku.edu.tr"
```

In line 1, a command to change directory is bookmarked at index 0 and the next command is bookmarked at index 1. Using *-l* lists all the bookmarks added to **shellington**. *bookmark -i idx* executes the command bookmarked at index *idx*. Using *-d idx* deletes the command at index *idx* and shifts the successive commands up in the list.

### remindme (15 pts)

Another command that you have to implement in **shellington** is called **remindme**. This command allows you to set up a reminder message that pops up at a specified time every day. You are required to use *crontab* to implement this command in the source code of **shellington**.

An example of the command is as follow. You can write down any reminder message that you want and set up the exact time the message should pop up on your screen.

```
bash$ remindme 9.45 "Time to go to the OS class."
```

We suggest you to explore *crontab* and *notify-send* utilities before implementing this command. You can learn more about *crontab* from here and *notify-send* from this page.

### Your awesome command (10 pts)

This command is any new **shellington** command of your choice. Come up with a new command that is not too trivial and implement it inside of **shellington**. Be creative. Selected commands will be shared with your peers in the class. Note that many commands you come up with may have been already implemented in Unix. That should not stop you from implementing your own. Also note that **if you are in a team of 2, each team member should implement one command each**.

## Part IV - Kernel Modules

### 30 points

In this part of the project, you will write two new kernel modules that will be triggered using commands from **shellington**. You need to be a superuser or have a sudo access in order to complete this part of the project. The shell commands that invoke the kernel modules are as follows.

- **pstraverse <PID> <-d or -b>**: The `pstraverse` command finds the subprocess tree by treating the given PID as the root and traverse the tree in depth-first-search (DFS) or breadth-first-search (BFS) order depending on the second command line argument, i.e. -d or b. If the value is -d, then the tree is traversed in DFS order. Otherwise, if the value is -b, the tree is traversed in BFS order.

  When a `task_struct` node in the subprocess tree is visited during the traversal, the PID and the name of the executable run by the process are printed using printk by the kernel module.

  The followings are some suggestions and details for your implementation.

  - You will need to explore the Linux task struct in `linux/sched.h` to obtain necessary information such as process id (PID) and name.
  - Test your kernel module first outside of **shellington** and make sure it works.
  - When the command is called for the first time, **shellington** will prompt sudo password to load the module into the kernel. After the module is loaded by the first call, the tree traversal operation on the targeted PID will run. Successive calls to the command will not load the module again. They will only invoke tree traversal operations by on targeted PIDs. In the first call of the command, the

traversal operation can be run by the initialization function of the kernel module. In the following calls, you can use ioctl function calls to trigger the operations.

  - **shellington** should remove the module from kernel when the shell is exited.
  - You can use `pstree` command to check if the process list is correct. Use -p to list the processes with their PIDs. Note that there might be some processes that are shown by pstree but not shown by your kernel module. These processes are the ones whose names are in curly brackets when printed by `pstree -p`.

- **filelist 'file_name'**: The `filelist` command creates a character device file named file_name and prints its name and the siblings of this file in /dev directory using printk. The siblings of a file 'file_name' here refer to the other files that are located in the same directory with 'file_name'.

  The followings are some suggestions and details for your implementation of this command.

  - You will need to explore the Linux file struct, i.e. `struct file` in `linux/fs.h` to obtain necessary information such as file name and the siblings of the file.
  - Test your kernel module first outside of **shellington** and make sure it works.
  - When the command is called for the first time, **shellington** will prompt sudo password to load the module into the kernel, and print the name of the created file and its siblings using printk. File name and the siblings of the file do not have to be printed by the initialization function of the kernel module. When you call the command for the first time, you can load the module using `insmod` and, then, *open* the created character device file in **shellington**'s source code. Successive calls to the command will notify the user that the module is already loaded, and will not do anything.
  - **shellington** should remove the module from kernel when the shell is exited.

**Useful References:**

- Info about task link list (scroll down to Process Family Tree)
- Info on `struct file` from here and here, and the source code.
- Linux Cross Reference.
- You can use **pstree** to check if the sibling list is correct. Use -p to list the processes with their PIDs
- Even though we are not doing the same exercise as the book, Project 2 - Linux Kernel Module for Listing Tasks discussion from the book might be helpful for implementing this part

**READ CAREFULLY**

You are required to submit the followings packed in a zip file (named your-username(s).zip) to Blackboard:

- .c source code file that implements the **shellington** shell. Please comment your implementation.
- .c sourse code file that implements the kernel module you developed in Part IV.
- Any supplementary files for your implementations (e.g. Makefile)
- A short REPORT briefly describing your implementation, particularly the new command you invented in Part III. You may include your snapshots in your report.
- Do not submit any executable files (a.out) or object files (.o) to Blackboard.
- Each team must create a **github repository for the project** and add TAs as the contributors (usernames are msasongko17 and readleyj). Add a reference to this repo in your REPORT. We will be checking the commits during the course of the project and during the project evaluation. This is useful for you too in case if you have any issues with your OS.
- Insufficient amount of detail in the REPORT or in the code repository will result in a 10 point penalty in the project grade.
- Selected submissions may be invited for a demo session. Note that team members will perform separate demos. Thus each project member is expected to be fully knowledgeable of the entire implementation.

GOOD LUCK and START EARLY